Available online www.unicrossjournals.com

JOURNAL OF CONTEMPORARY RESEARCH (JOCRES)

RESEARCH ARTICLE VOL. 3 (2) ISSN:2814-2241

Date Published: June 30, 2024

Pages 129-149

REVIEWING THE SOFTWARE COST ESTIMATION PARADIGMS

Duke S. O.

Department of Computer Science, University Cross River State

Orokduke2003@yahoo.com & orokduke2003@unicross.edu.ng

Abstract

One of the most important activities of a software development project is project planning. A poorly planned software development project often results in software failure, which is always a nightmare for the project team. Today, the entire software engineering and software cost estimation communities are cognizant of the rise of software failures due to inaccurate or imprecise estimates. Several software cost estimation paradigms exist in the form of algorithms and non-algorithms, and a few number of them are presented in this paper. This paper further discusses software cost estimation model performance in terms of magnitude relative error. Magnitude relative error is a cost estimation model error measure criterion used for measuring the model's accuracy. An example of effort and duration estimations using COCOMO II and their values compared to the actual values of the software project was carried out.

Keywords- Software accuracy; COCOMO; COCOMO II; Software failure

.

1.0 Introduction

Generally, the actual cost, which has financial implications, depends on the effort needed to develop the software. The effort, in turn, depends on the size of the software. Size is the most important input for software development. Therefore, the effort accuracy, which is the most important, and the development cost of any software hangs on the prediction accuracy of the software size, which is not always known until the end of the software's development. Generally, these two factors or characteristics are very difficult to predict during a software development project. One of the reasons for this difficulty is that software development projects are not pure science and are often not unique, as most people have no previous experience or background with them. This makes the predictions of effort and cost seem complicated. One other reason is that a software development project is not a tangible task, so measuring effort, cost, and the amount of progress or milestones in the project is very difficult. In addition, the requirements for software development projects are dynamic and hence are changing continuously, and as such, the effort and cost estimations will continue to change. A lot of managers of software development projects usually try to avoid the use of effort or cost estimations, or sometimes do the estimations within a limited domain, because of the problems mentioned. It is necessary to recount or describe the importance and usage of software estimation paradigms or techniques and their effects on the success of the software project so that the software project managers become aware of their usage. Many problems in software development projects may be caused by inaccuracies in effort and cost estimations due to optimistic or pessimistic estimations. for the readers to have a better understanding of the software cost estimation paradigms, the specific objective of this paper is to illustrate their capacities and group them based on their attributes or characteristics.

2.0 Background

In the last few decades, software project failures have been an important topic. A lot of software failures occur during the planning and estimation stages; they usually don't occur during the implementation stage. According to Molokken and Jorgenson (2003), in spite of the cost and overrun time, approximately between 30% and 40% of the software development projects are concluded, and the rest fail. Glass (2006) reports that The Standish Group's CHAOS reports a failure rate of 70% for the software projects. Jørgensen and Molokken-Ostvold (2006) report that the cost overrun was indicated at 89% in the 1994 CHAOS. According to Glass (2006), the reported results do not depict the real failure rate and are pessimistic. Furthermore, Jørgensen and Moløkken-Ostvold (2006) pointed out that the CHAOS report may be corrupted. Notwithstanding, the statistics mentioned indicate a deep crisis pertaining to the future of the software projects (Jørgensen and Molokken-Ostvold, 2006). In fact, in the last few decades, several studies have been carried out to find the reasons for software project failures. A study indicates that Galorath and Evans (2006) carried out an in-depth search of 2100 internet sites and discovered 5000 reasons for the software project's failures. Among the reasons they found, poor planning of the software project, insufficient requirements engineering, inaccurate estimations, and sudden decisions at the early stages of the project were the most important reasons. Other research regarding the reason for project failures shows that inaccurate estimation is the root factor in most software project failures (Kemerer, 1987; Moløkken and Jørgensen, 2003; Jørgensen, 2005; Jones, 2007). Although the software project statistics indicated, perhaps pessimistically, that inaccurate software cost estimation is a real problem in the world of software engineering, it should be solved. Presenting efficient methods and reliable software cost estimation models seems necessary regarding the mentioned problem. The software projects are not stable, and their state will be on a continuum, so many techniques should be presented for estimating whether each technique is proper for a particular project or not.

3.0 Estimation techniques

Many techniques for software cost estimation exist. These techniques are grouped into two categories, namely, algorithmic and non-algorithmic techniques. The algorithmic techniques are based on mathematical formulas, while the non-algorithmic techniques are non-formula-based techniques. For accurate software cost estimation, both groups may be required. If we know the requirements better, their performance will be better. In this subsection, some of the most popular estimation techniques are discussed.

3.1 Algorithmic Software Cost Estimation Paradigms

The algorithmic software cost estimation paradigms are based on a special algorithm. They usually need data at first and make results by using mathematical formulas or equations. Today, several software estimation techniques use these models.

The algorithmic software cost estimation paradigms are categorized into some different models. Each algorithmic model uses an equation to estimate the cost of developing software in the form shown in equation 1:

$$Effort = \text{f}\,(\text{w1}, \text{w2}, \ldots, \text{wn}) \quad \ldots \quad (1)$$

where (y1, w1, …, wn) is the vector of the cost factors or cost drivers. The differences among the existing algorithmic methods are related to choosing the cost factors and functions. All the cost factors used in these models are:

- Product factors: needed reliability, product complexity, database size used, required reusability, and documentation matching life-cycle needs.
- Computer factors: main storage constraint, execution time constraint, computer turnaround constraint, and platform volatility.
- Personnel factors: capability of the analyst, programming capability, language and tool experience, application experience, platform experience, and personnel continuity.
- Project factors: multisite development, use of software tools, required development schedule.

Quantifying the factors mentioned is very difficult to do, and some of them are ignored in some software projects. In this study, several algorithmic methods are considered the most popular. The methods mentioned have been selected based on their reputation. There are many papers that use the selected algorithmic methods (Musilek, Pedrycz et al., 2002; Yahya, Ahmad et al., 2008; Lavazza and Garavaglia, 2009; Yinhuan, Beizhan et al., 2009; Sikka, Kaur et al., 2010).

When computing the Size Lines Of Codes (SLOC) for software, its amount is compared with other previous software projects whose size lines of codes have been computed, and the size of the project is estimated. SLOC measures the size of the project easily. When the software project has been completed, all estimations are compared with the actual ones.

On a larger scale, Thousand Lines of Code (KLOC) are used for the estimation of the cost of developing software. The use of the KLOC metric is very common in a lot of estimation methods. At the early stages, SLOC measurement of the project appears to be very difficult because of a lack of information about requirements.

Comparison of the size of software that deploys different languages is very hard because SLOC is computed based on the language instructions; comparing the size of software that uses different languages is too hard. For many complicated software estimation techniques, SLOC is the basis for the estimation models. According to Pressman (2005), SLOC is normally computed by considering the Lowest Size as (SL), the Highest Size as (SH), and the Most probable Size as (SM) as the most probable size. Their relationship is expressed as:

S = SL + 4SM + SH

- **Source Line of Code**

SLOC is a prediction parameter that shows how many commands and data definitions there are, but it doesn't include instructions like comments, continuation lines, or blanks. It's usually used as an example based on a technique for estimation.

1. Linear Models

This is the commonly used linear model with the simple structure shown Equation 2.

$$\boldsymbol{Effort = a_0 + \sum_{i=0}^{n} a_i x_i} \qquad (2)$$

$a_1, a_2, \ldots, a_n$ are always selected according to the project's information.

2. Multiplicative models

This model has the form shown in Equation 3.

$$Effort = a_0 * \prod_{i=1}^{n} {a_i}^{x_i} \qquad (3)$$

In this case, $a_1, \ldots, a_n$ are selected according to the project's information. The allowed values for x_i are -1,0 and +1 for this model.

3. Function Point Size Estimates

Albrecht introduced the Function Point (FP) metric in 1983 to measure the functionality of a software project. Basically, this technique of size estimation involves the identification of the indicators listed below:

- User Inputs,
- User Outputs,
- Logic files,
- Inquiries,
- Interfaces

In this technique, a Complexity Degree is defined for each indicator, which lies between 1 and 3. 1 is simple, 2 is medium and 3 is complex. Each indicator also has a weight that can range from 3 to 15.

The first step is to count the number of each of the indicators mentioned, and then the complexity degree and the weight are multiplied together. In general, the Unadjusted Function Points Counts (UFPC) is defined as follows:

$$UFPC = \sum_{i-1}^{5} \sum_{j-1}^{3} N_{ij} W_{ij} \qquad (4)$$

where Nij accounts for the number of indicators i with complexity j and Wij accounts for the weight of indicator i with complexity j.

Previous experience suggests that FP may be useful for software estimation due to its ability to be calculated on the basis of requirement specifications at an early stage of the project. To calculate the FP, UFPC should be increased by multiplying it by a factor called a Technical Complexity Factor (TCF) which must be derived from the elements or components shown in Table I.

TABLE 1: Technical Complexity Factor elements

F1	Reliable back-upand recovery	F8	Data communications
F2	Distributed functions	F9	Performance
F3	Heavily used configurati on	F10	Online data entry
F4	Operational ease	F11	Online update
F5	Complex interface	F12	Complex processi ng
F6	Reusability	F13	Installation ease
F7	Multiple sites	F14	Facilitate change

Each component can have a value that lies between 0 and 5. A value of 0 means the component has no impact on the project. A value of 5 means the component is mandatory and very important. The final total TCF for the software project is then computed using equation 5.

$$TCF = 0.65 + 0.01 * \sum Fi \qquad (5)$$

TCF ranges from 0.65 if all Fi are 0 to 1.35 if all Fi are 5. Finally, FP is computed using Equation (6).

$$FP = UFPC * TCF \qquad (6)$$

4. Software Evaluation and Estimation of Resources - Software Estimating Model Seer-Sem

Software Evaluation and Estimation of Resources - Software Estimating Model (Seer-Sem) was proposed and formulated by Galorath in 1980 (Galorath, 2006). The majority of the parameters in this technique are commercial, and business software projects normally use SEER-SEM as their main estimation technique. The most important feature or attribute of this technique is the size of the software. In this method, the required size is always computed as the effective size (Se) and is always computed using five indicators, namely: Existingsize, Newsize, reimpl, and retest as shown in Equation 7.

$$Se = Newsize + Existingsize(0.4Redesign + 0.25reimp + 0.35\ Retest \qquad (7)$$

When the computation of Se is concluded, its value now forms part of the input for the required effort for software development. The required effort is thus estimated using Equation 8.

$$Effort = D^{0.4} * \left(\frac{Se}{C_{te}}\right)^{E} \quad (8)$$

where D is relevant to the personnel aspects; its determination is based on the complexity degree in the personnel's structure. Cte is calculated based on the productivity and efficiency of the project technique used in the commercial projects. (Fischman et al., 2005).

5. COCOMO

Software Cost Estimation Models generally use some form of cost indicator for the estimation of software development. Constructive Cost Model (COCOMO).

Barry Boehm introduced the concept of COCOMO in 1981. Since then, various versions of the concept have been developed by Boehm. COCOMO has been the most popular and widely used technique for software cost estimation. COCOMO is an algorithmic method that uses some parameters and equations that have been derived from past experiences with software projects for estimation.

The latest version of the COCOMO is COCOMO II. This version of COCOMO estimates the amount of effort required to develop software in Person-Month (PM).

COCOMO II uses lines of code or function points as the size metrics and 17 effort multipliers generated from cost drivers (Table 2) and five scale factors (Table 3). Several rating levels are associated with scale factors, such as very low (very low), low, nominal (nominal), high (very high) and extra high (extra high). Each rating level is assigned a numerical value as its weight.

COCOMO II possesses a number of unique characteristics that differentiate it from other methods. The scope of the application of COCOMOs is extensive, and the outcomes are typically reliable.

When estimating the effort required to develop software and the duration or schedule, equations 9 and 10 are used, respectively.

TABLE 2: Effort Multipliers

Attributes	Type	Description
RELY	Product	Required system reliability
CPLX	Product	Complexity of system modules
DOCU	Product	Extent of documentation required
DATA	Product	Size of database used
RUSE	Product	Required percentage of reusable components
TIME	Computer	Execution time constraint
PVOL	Computer	Volatility of development platform
STOR	Computer	Memory constraints
ACAP	Personnel	Capability of project analysts
PCON	Personnel	Personnel continuity
PCAP	Personnel	Programmer capability
PEXP	Personnel	Programmer experience inproject domain
AEXP	Personnel	Analyst experience in project domain
LTEX	Personnel	Language and tool experience
TOOL	Project	Use of software tools
SCED	Project	Development schedule compression
SITE	Project	Extent of multisite working and quality of inter-site communications

TABLE 3: Scale factors

Factor	Explanation
Precedentedness (PREC)	Reflects the previous experienceof the Organization
DevelopmentFlexibility (FLEX)	Reflects the degree of flexibilityin the development process.
Risk Resolution (RESL)	Reflects the extent of riskanalysis carried out.
Team Cohesion(TEAM)	Reflects how well the development team knows eachother and work together.
Processmaturity (PMAT)	Reflects the process maturity ofthe organization.

$$E = A * (Size)^{Z} * \prod_{i=1}^{17} EM_i \qquad (9)$$

Where E = effort; A = 2.94; B = 0.91

$$Z = B + 0.01 * \sum_{j=1}^{5} SF_j$$

EM = effort multiplier (for i = 1, 2, …, 17); SF = scale factor (for j = 1, 2, …,5)

$$TDEV = C * (E)^{K} \qquad (10)$$

Where C = 3.67; D = 0.28

$$K = D + 0.2 * 0.01 * \sum_{j=1}^{5} SF_j$$

6. Putman's SLIM model

SLIM stands for Software Lifecycle Management. SLIM was proposed by Putman based on manpower distribution and the examinationof many software projects (Kemmerer, 2008). This model is one of the earliest software cost estimation models developed, and also among the most widely used model. Equation 11 shows the Putnam's SLIM model for the estimation size metrics.

$$S = C * (Effort)\wedge\frac{1}{3} * (t_d)\wedge\frac{4}{3} \qquad (11)$$

C represents a technology factor and environmental indicator that shows environmental ability. C is a composite cost driver involving 14 components.

t_d is the time needed to finish and deliver the software in years.

S denotes the size of the software in lines of code.

Putnam furthermore present another formula for computing the Manpower-buildup factor shown in Equation 12.

$$D = \frac{E}{t^3} \qquad (12)$$

D: is called the manpower acceleration constant;

D is 12.3 for new software having many interfaces and interactions with other systems, and 15 for standalone systems. D is 27 for reimplementation of the existing systems. Furthermore, D is said to vary from 8 for new software to 27 for rebuilt software.

E represent the total software development effort required to build the software in terms of person-year and

t is the number of years it will take to complete the software development.

To estimate the effort required to develop software Putnam presented another equation for the estimation of effort shown in equation 13 after combining Equations 11 and 12.

$$Effort = \frac{S}{C^{\frac{9}{7}}} * D^{\frac{4}{7}} \quad (13)$$

3.2 Non-Algorithmic Software Cost Estimation Paradigms

Unlike the Algorithmic paradigms, the methods of this group rely on analytical comparison and inference. In order to use the non-algorithmic paradigms, some information about previous projects that are similar to the one under estimation will be needed, and usually the estimation process in these methodologies is based on the analysis of previous datasets. Here, we have selected three methods for the assessment because these are more widely used than the other non-algorithmic methods, and many papers on their use have been published in recent years (Jorgensen, 2009; Xie, 2007; Idri, 2004; Braz, 2006; Keung, 2008; Mbarki, 2006; Li, 2008; Jianfeng, 2009; Pahariya, 2009; Ravi, 2009; Cuadrado, 2009; Rodri, 2010; Jianfeng, 2009; Lin, 2008; Kitchenham, 2008).

1. Expert Judgment

In order to make an estimation based on expert judgment, it is often necessary to obtain advice from experts who have extensive experience in the same type of project. This technique is typically employed when there is a lack of data or requirements to be met. The fundamental issue in this approach is consultation. Delphi is one of the most commonly used methods that works in accordance with this technique. Delphi organizes a special meeting between project experts and attempts to gain a true understanding of the project through their discussions or debates. The steps which Delphi includes are:

i. The coordinator or head of the team gives each expert with an estimation form to fill.
ii. Without any discussion with others, each expert presents his own estimation.
iii. The coordinator gathers or collates all the forms and sums them up, finds mean or

median on a form and ask the experts to start another iteration.

iv. Steps ii and iii are repeated or iterated until an agreement is reached on an estimate before stopping.

$$Sim(p, p') = \frac{1}{[\sqrt{\sum_{i=0}^{n} w_i \, Dis(f_i, f'_i)} + \delta]} \qquad (15)$$

$$\delta = 0.0001$$

$$Dis(f_i, f'_i) = \begin{cases} |f_i - f'_i|, if\ f_i\ and\ f'_i\ are\ numeric\ or\ ordinal \\ 1 \qquad if\ f_i\ and\ f'_i\ are\ nominal\ and\ f_i = f'_i \\ 0 \qquad if\ f_i\ and\ f'_i\ are\ nominal\ and\ f_i \neq f'_i \end{cases}$$

Mahmud S et al. (2008) demonstrates an example of deploying the Delphi technique in which eight experts made their contributions after going through four stages and the final convergence or a consensus was determined. This example is shown in Figure 1.

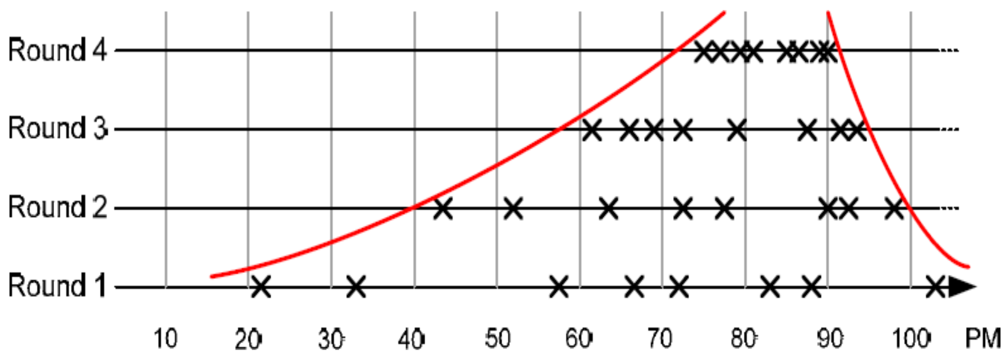

Figure1: An example of using Delphi by eight experts

2. Analogy

The analogy technique, also known as case-based reasoning, involves observing a number of similar software projects that have been completed and then estimating effort and cost based on their actual costs and effort. The analogy estimation is made at both the total system and subsystem levels. By examining the results of the previous actual projects, it is possible to estimate the effort and cost of a similar software project. To actualize software cost estimation when using this technique, the following algorithm is adopted:

i. Choose the analogy technique.
ii. Investigate the similarities and differences of previous and the target software.
iii. Examine the analogy quality.
iv. Provide the software cost estimation.

According to Shepperd & Schofield (1997) and Chiu & Huang (2007), two popular similarity functions exist, namely Euclidean Similarity (ES) and Manhattan (MS), for similarity comparison of similar software. These functions define similarity to compare the features of two software.

$$Sim(p, p') = \frac{1}{[\sqrt{\sum_{i=0}^{n} w_i \, Dis(f_i, f'_i) + \delta}]} \qquad (14)$$

$$\delta = 0.0001$$

$$Dis(f_i, f'_i) = \begin{cases} (f_i - f'_i)^2, & if\ f_i\ and\ f'_i\ are\ numeric\ or\ ordinal \\ 1 & if\ f_i\ and\ f'_i\ are\ nominal\ and\ f_i = f'_i \\ 0 & if\ f_i\ and\ f'_i\ are\ nominal\ and\ f_i \neq f'_i \end{cases}$$

where

Sim stands for similarity; D is stands for distant;

p and p' are projects; w_i is the weight assigned to each feature and varies between 0 and 1. f_i and f_i'display the ith feature of each software; and n illustrates the number of features. δ is used to obtain the none zero results. The formula for MS is very similar to that of ES but it calculates the absolute difference between features as shown in Equation 15.

3.3 Machine learning Models

The majority of software cost estimation methods use statistical approaches to arrive at their results. Their results are not very strong and are not able to present reason as they are deterministic-based results. Machine learning techniques, on the other hand, could be suitable in software cost estimation because they can increase the accuracy of software cost estimation via the training rules of estimation and repeating the run cycles. The results produced by the machine learning paradigms are stochastic-based; hence, they are very strong and able to present reason. We can classify the machine learning paradigms into two categories, which include Fuzzy Logic (FL) and Neural Networks (NN).

The FL Technique

The FL technique is a type of technique that tries to mimic or simulate the behavior of a human being. Every system that adopts the FL technique does so in order to be developed through simulation of human behavior and reasoning. FL is a candidate and an efficient tool for solving problems that involve difficult decision-making under vague conditions. The steps for implementing FL are shown in the following algorithm:

Step 1: Fuzzification: captures and fuzzifies crisp inputs (converts crisp inputs into fuzzy inputs).

//Determines the degree of membership in the rightful fuzzy sets via the membership function.

Step 2: Rules construction and evaluation rules are constructed in the antecedent and consequent sections using IF...THEN statements.

//The fuzzy inputs generated in step 1 are used to construct the linguistic FL rule statements.

//Typically, a FL rule has two parts: the IF... part in the antecedent or premise and the THEN part in the consequent or conclusion.

//The rule constructions and interpretations take place within the knowledge-based and fuzzy inference engines. The knowledge-based component stores the fuzzy rules, while the fuzzy inference engine interprets them.

// This section of the FL system converts the fuzzy inputs into fuzzy outputs.

Step 3: FL Implication defines what happens with the FL operators "OR" and "AND" used in constructing the fuzzy rules.

//FL operators are used to evaluate multi-criteria, multi-value or multi fuzzy set in decision-making problem.

// The work of FL implication in the fuzzy inference system (fis) is to modify the fuzzy outputs.

//Two types of implications methods exist in the FL system, namely truncation (or clipping) and scaling (or squashing).

//Truncation implication adopts the min-function method to modify the fuzzy outputs, while the scaling implication adopts the prod-function.

Step 4: Aggregation

//Merges or unifies the fuzzy outputs of every rule into a single fuzzy output constructed within the concerned fuzzy system.

Step 5: Defuzzification is the last operation within the fis.

//Converts a fuzzy (imprecise) output into a precise output value.

//Calculates the effort needed to develop software development tasks and to correlate the existing software cost technique.

The NN Technique

NNs consist of layers, and each layer is made up of several elements called neurons. Through the weights defined for the inputs, neurons create the outputs. The outputs, therefore, are the actual effort and hence the primary objective of estimation.

The Back Propagation Neural Network (BPNN) is an optimal choice and the most suitable solution for software estimation problems due to its ability to modulate and adjust the weights by comparing or correlating the network outputs with real-world outcomes or actual results. Furthermore, the training process is highly efficient.

4.0 Integration of FL and NN with Software Cost Estimation Model

Any software cost estimation model can be integrated with FL and NN techniques to enhance its accuracy in estimating the cost of developing software.

4.1 COCOMO II-Fuzzy Logic (COCOMO II-FL) Integration

According to Attarzadeh, (2010), COCOMO has been implemented with FL technique. Figure 2 demonstrates how the two methods have been combined to estimate the effort required to engineer the software.

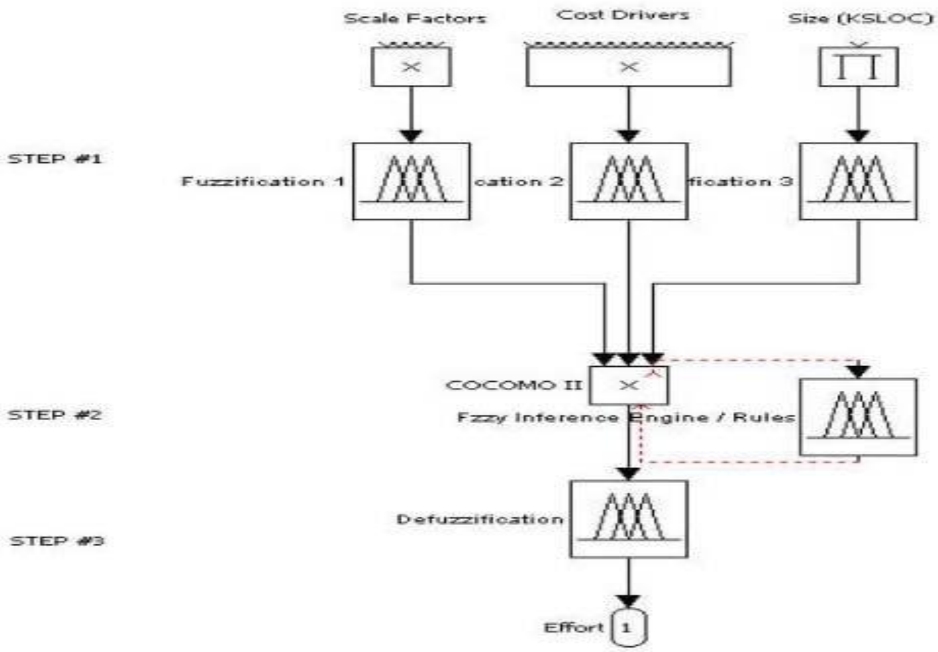

Figure 2: COCOMO II – Fuzzy Logic Integration

In carrying out the process of software cost estimation by means of COCOMO-FL, all the steps mentioned in the algorithm in the FL technique were applied. Figure 2 shows the three main interfaces of the FL implementation: fuzzification, fuzzy inference engines/rules. The first activity involved fuzzification of the crisp inputs—cost drivers, scale factors, and size of software. The second activity is the construction of fuzzy rules within the knowledge-based section and the interpretation of the rules by the fuzzy inference engine. The next and last step is the defuzzification of the outputs of step two to provide the effort required to develop the software.

4.2 COCOMO II-Neural Network (COCOMO-NN) Integration

Many of the studies on the utilization of NNs for estimating the cost of software development are centered around modeling the COCOMO method. For instance, Attarzikadeh (2010) asserts the proposal of NN for estimating the cost of software development in terms of effort. The framework for COCOMO-NN implementation is shown in Figure 3. Furthermore, Figure 3 also shows the layers, inputs – Effort Multipliers (EMs), Scale Factors (SFs), and the transfer function of the NN. Shown also in Figure 3 are the weights Pi and Qj for SFs and EMs, respectively. In real sense, the inputs to the NN are the weight of the EMs and the weight of the SFs.

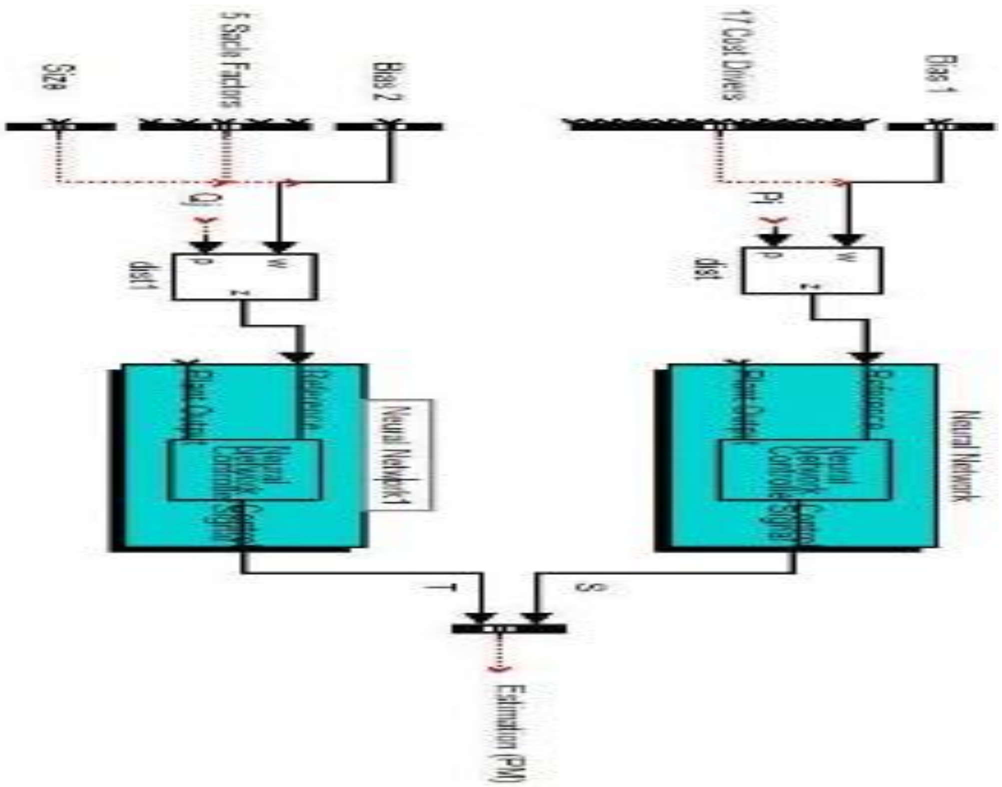

Figure 3: COCOMO II – Neural Network Integration

5.0 HINT AND METHOD FOR COMPARING THE ESTIMATION TECHNIQUE

This section provides the hint and methods of comparing the performance of software cost estimation techniques based on the previous subjects presented. Again, the comparison is based on the advantages and disadvantages of these techniques. The comparison is very important because it could be beneficial for choosing an appropriate software cost estimation technique for a particular software development project. But then, selecting the software cost estimation technique is usually a hard thing to do, and one needs to have a lot of data about the current state of the software development project. With the availability of enough data, these techniques offer a very reliable result. Moreover, the algorithmic techniques are usually very complementary to each other. For instance, COCOMO II uses the Function Point and Size Lines Of Code (SLOC) as two input metrics, and generally, if these two size metrics are accurate, then COCOMO II will also provide accurate results. Finally, choosing the best method to estimate the cost of developing software and viewing the available information about the current project's data and that of the similar previous or existing software projects could be beneficial. Table 4 presents the comparison of the existing software cost estimation techniques.

TABLE 4: Comparison of the existing Software cost Estimation Techniques

Method	Type	Advantages	Disadvantages
COCOMO II	Algorithmic	Provides very common and clear results	Not suitable for all projects and requires much data.
Function Point	Algorithmic	It is a language free technique. Provides better results than SLOC would	Quality of the output is not considered as mechanization is difficult to do.
Expert Judgment	Non-Algorithmic	Provides fast predictions.	The success of this method depends on the expert, most time is done incomplete
Analogy	Non-Algorithmic	New projects are based on actual experiences, having especial expertise not important	Much of the information about the past projects is required. In some cases, similar projects are hard to come by.
FL	Non-Algorithmic	It is Flexible and data training is not needed.	Difficult to use and the degree of meaningfulness is difficult to maintain.
NN	Non-Algorithmic	Consistent with unlike databases, Power of reasoning	Performance depends on large training data and does not need any guideline for designing.

6.0 EVALUATING THE RESULTS OF ESTIMATION METHODS OR MODELS

It is very necessary to carry out performance evaluations on software estimation models. This approach enables us to discover the enhancements in one model over the others. The acceptability of the use of these software cost estimation techniques was a very big issue decades ago, and so there was room for evaluating their performance using software estimation model error evaluation metrics.

Quite a number of metrics for measuring the performance of software cost estimation techniques are available. In this subsection, the popular ones are presented. These are Magnitude of Relative Error (MRE), Mean Magnitude of Relative Error (MMRE), and Percentage Relative Error Deviation (PRED). The quality or accuracy of any software cost estimation model depends on these model accuracy criteria or metrics. These criteria or metrics are the most widely used evaluation measures or methods to evaluate the performance accuracy of any software cost estimation model in the software engineering domain. These are shown in Equations 16, 17, and 18, respectively.

The MRE criterion is the value error estimated for each of the projects compared to the actual. This is computed using Equation 16.

$$MRE_i = \left(\left| \frac{estimated\ effort_{(i)} - actual\ effort_{(i)}}{actual\ effort} \right| \right) \quad \ldots 16$$

The MMRE criterion is used to compute the error of the mean value of the MRE_i for the project. It computes for a given project the difference between the actual and the estimated effort relative to the actual effort. This is calculated by using Equation 17.

$$MMRE = \frac{1}{n}\sum_{i=1}^{n} MRE_i \quad \ldots 17$$

The PRED criterion is used to estimate the accuracy of the estimation models. The typical PRED model is shown in Equation 18.

$$\boldsymbol{PRED(x) = \frac{1}{n}\begin{cases} 1, & if MRE_i \leq x \\ 0, & otherwise \end{cases} \ldots 18}$$

Where x = 0.25 and n = numbers of project.

7.0 ACTUAL ESTIMATION USING COCOMO II

In this subsection, a demonstration of a software development effort estimate is carried out using COCOMO II. COCOMO II being the most popular technique, its metrics are used for real software cost estimation projects. The cost drivers (effort multipliers) and scale factor of COCOMO II as well as their weights shown in Tables 5 and 6 are correlated or tied up to real-life software development projects. According to Maged et al. (2008), data on about thirty software development projects has been gathered from different organizations, which include banking, insurance, web development, communication, etc., via distributing the questionnaire.

Only one project data out of the thirsty projects recorded in the dataset is used to illustrate the execution capabilities of COCOMO II.

Depending on the individual manager, maximum and minimum values for the key or main cost drivers may be assigned to demonstrate how they affect or influence the effort estimate. For example, maximum values for cost drivers will result in an effort estimate that is more than three times the original estimate, while low values will reduce the effort estimate to approximately one-third of the original estimate. In this study, we have chosen the cost drivers and scale factors shown in Tables 5 and 6 as our key cost drivers and scale factors. We also chose their ratings and weights, or values, between low, nominal, high, and very high. assigned maximum values or weights to them.

TABLE 5: Randomly selected Cost Drivers

Cost Driver	Rating	Value
RELY	Very High	1.26
DATA	Very high	1.28
CPLX	High	1.17
RUSE	Very High	1.15
DOCU	Nominal	1.00
TIME	High	1.11
STOR	High	1.06
PVOL	Low	0.87
ACAP	Very high	0.71
PCAP	Very High	0.76
PCON	Very high	0.81
AEXP	Very high	0.81
PEXP	Very High	0.85
LTEX	Very High	0.84
TOOL	Very High	0.78
SITE	Very High	0.86
SCED	Nominal	1.00

8.0 Computation of effort using randomly selected cost drivers and scale factors

Let us now compute the effort required to develop software of size 130 Thousand Source Lines of Codes (KSLOC) using randomly selected cost drivers and scale factors shown in Tables 5 and 6. The real effort for this software development is 117.92 person- month and the duration is 15.5 months.

TABLE 6: Randomly selected Scale Factors

Scale Factor	Rating	Value
PREC	High	2.48
FLEX	Nominal	3.04
RESL	High	2.83
TEAM	Very High	1.10
PMAT	Very High	1.56

Using the cost drivers and scale factors shown in Tables 5 and 6, and Equations 9, 10, and 16, the effort, TDEV, and MRE are estimated as follows:

$$\text{Effort} = 2.94 * (130)^{0.91+0.01*11.01} * 0.377 = 143.36 \text{ pm}$$
$$\text{TDEV} = 3.67*(143.36)^{0.28+0.2*0.01*11.01} = 16.44 \text{ months}$$
$$\text{MRE} = (143.36 - 117.92)/117.92 = 0.22$$

As can be seen from the computation, the differences between the estimated effort and the original effort are very small. Also, the estimated duration, TDEV, is almost the same number of months as the real. Finally, the MRE, which is 0.22, is a very reasonable value within the benchmark of 0.25 for model error. This demonstrates the ability of the COCOMO technique to be applicable to software cost estimation. Hence, COCOMO technique metric results accuracy can be increased via calibration.

9.0 CONCLUSION

Software sometimes fails; hence, unearthing or discovering the reasons for these failures has been the topic of a lot of research for many years now. The root or underlying cause of software failures is an inaccurate or imprecise estimate, which most often occurred in the early phase of software development, as presented in this study, according to a number of studies. Therefore, to achieve accurate and reliable software development estimates, it is pertinent and very important to introduce and focus attention on software cost estimation techniques. Saddled with responsibility, software managers are used to choosing the software cost estimation technique they consider to be the best based on the nature and conditions of the software project at hand. In doing this, it goes a long way toward reducing software failures. It is agreed that none of the software cost estimation techniques is capable of providing the best estimates, as all of them are not suitable for all software projects. Some are suitable for specific software projects. Choosing the best software cost estimation technique depends on how its principles and application are understood. Each software cost estimation technique's performance is a function of many parameters, which include the development method, complexity of the software project, personnel expertise, the duration of the software project, etc. In this paper, a few performance evaluation metrics are accounted for. A description of the performance of COCOMO II is also presented in this paper by comparing it with an actual or real example.

References

[1] Attarzadeh, I. Siew Hock Ow, "Proposing a New Software Cost Estimation Model Based on Artificial Neural Networks", IEEE International Conference on Computer Engineering and Technology (ICCET), Volume: 3, Page(s): V3- 487 - V3-491 2010.

[2] Attarzadeh, I. Siew Hock Ow, "Improving the accuracy of software cost estimation model based on a new fuzzy logic model", World Applied science journal 8(2):117-184,2010-10-2.

[3] Albrecht. A.J. and J. E. Gaffney, "Software function, source lines of codes, and development effort prediction: a software science validation", IEEE Trans Software Eng. SE, pp.639-648, 1983.

[4] Braz, M. R. and S. R. Vergilio. "Software Effort Estimation Based on Use Cases". Computer Software and Applications Conference, COMPSAC '06. 30th Annual International, 2006.

[5] Boehm, B. W., & Valerdi, R. "Achievements and challenges in cocomo-based software resource estimation", IEEE Software, 25(5),74- 83.doi:10.1109/MS ,2008.

[6] Boehm, "Software Engineering Economics", Prentice Hall,1981.

[7] Chiu, N.H., Huang, S.J., "The adjusted analogy- based software effort estimation based on similarity distances", Journal of Systems and Software 80 (4), 628–640.2007.

[8] Cuadrado-Gallego, J. J., Rodri, et al. "Analogies and Differences between Machine Learning and Expert Based Software Project Effort Estimation". Software Engineering Artificial Intelligence Networking and Parallel/Distributed Computing (SNPD), 11th ACIS International Conference on 2010.

[9] Fischman, L, K. McRitchie, and D. D. Galorath, "Inside SEER-SEM", CrossTalk, The Journal of Defense Software Engineering, 2005.

[10] Galorath, D. D., & Evans, M. W. "Software sizing, estimation, and risk management: When performance is measured performance improves". Boca Raton, FL: Auerbach, 2006.

[11] Glass, Robert L. (2006). The Standish Report: Does It Really Describe a Software Crisis? https://www.umsl.edu/~sauterv/analysis/Standish/glass.pdf

[12] Idri, A., S. Mbarki, et al. "Validating and understanding software cost estimation models based on neural networks". Information and Communication Technologies: From Theory to Applications, 2004. Proceedings. 2004 International Conference on 2004.

[13] Jones, C. "Estimating software costs: Bringing realism to estimating (2nd ed.)". New York, NY: McGraw-Hill, 2007.

[14] Jianfeng, W., L. Shixian, et al. “Improve Analogy-Based Software Effort Estimation Using Principal Components Analysis and Correlation Weighting”. Software Engineering Conference, 2009. APSEC '09. Asia-Pacific, 2009.

[15] Jorgensen, M., B. Boehm, et al. "Software Development Effort Estimation: Formal Models or Expert Judgment?" Software, IEEE 26(2): 14- 19,2009.

[16] Jørgensen, M. “Practical guidelines for expert-judgment-based software effort estimation”, IEEE Software, 22(3), 57-63. doi:10.1109/MS. 73, 2005.

[17] Keung, J. W., B. A. Kitchenham, et al. “Analogy- X: Providing Statistical Inference to Analogy- Based Software Cost Estimation”. Software Engineering, IEEE Transactions on 34(4): 471-484,2008.

[18] Kemerer, C. “An empirical validation of software cost estimation models”, Communications of the ACM, 30(5), 416-429. doi: 10.1145/22899. 22906, 1987.

[19] Lavazza, L. and C. Garavaglia “Using function points to measure and estimate real-time and embedded software: Experiences and guidelines”. Empirical Software Engineering and Measurement ESEM 2009. 3rd International Symposium on 2009.

[20] Li, J., J. Lin, et al. “Development of the Decision Support System for Software Project Cost Estimation. Information Science and Engineering”, 2008. ISISE '08. International Symposium on 2008.

[21] Li, Y. F., M. Xie, et al. “A study of genetic algorithm for project selection for analogy-based software cost estimation. Industrial Engineering and Engineering Management”, IEEE International Conference on 2007

[22] Mahmud S. Alkoffash, Mohammed J. Bawaneh and Adnan I. Al Rabea, “Which Software Cost Estimation Model to Choose in a Particular Project”, Journal of Computer Science 4 (7): 606- 612, 2008.

[23] Maged A. Yahya, Rodina Ahmad, Sai Peck Lee, “Effects of Software Process Maturity on COCOMO II’s Effort Estimation from CMMI Perspective”, 978-1-4244-2379-8/08 IEEE (c), 2008.

[24] Musilek, P., W. Pedrycz, et al. “On the sensitivity of COCOMO II software cost estimation model”. Software Metrics, Proceedings. Eighth IEEE Symposium on 2002.

[25] Moløkken, K., & Jørgensen, M.” A review of surveys on software effort estimation. International Symposium on Empirical Software Engineering”, 223-231. Retrieved from ACM Digital Library database, 2023.

[26] Shepperd, M., Schofield, C, “Estimating software project effort using analogies” IEEE Transactions on Software Engineering 23 (12), 733–743.1997.

[27] Pressman, Roger S. “Software Engineering: A Practioner's Approach”, 6th Edn., McGraw-Hill New York, USA., ISBN: 13: 9780073019338, 2005

[28] Pahariya, J. S., V. Ravi, et al. "Software cost estimation using computational intelligence techniques". Nature & Biologically Inspired Computing, NaBIC

2009. World Congress on, 2009.

[29] Sikka, G., A. Kaur, et al. “Estimating function points: Using machine learning and regression models". Education Technology and Computer (ICETC), 2nd International Conference on 2010.

[30] Yahya, M. A., R. Ahmad, et al. “Effects of software process maturity on COCOMO II’ effort estimation from CMMI perspective". Research, Innovation and Vision for the Future, RIVF. IEEE International Conference on 2008.

[31] Yinhuan, Z., W. Beizhan, et al. “Estimation of software projects effort based on function point". Computer Science & Education. ICCSE. 4th International Conference on 2009.